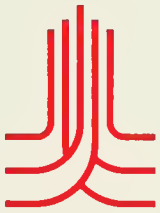
Modifying Two-Phase Locking to Improve Performance

by

Theodore Johnson

Ultracomputer Note #133

April, 1988



Ultracomputer Research Laboratory

NYU UCN-133
Johnson, Theodore
Modifying two-phase locking
to improve performance.
c.1

New York University
Courant Institute of Mathematical Sciences
Division of Computer Science
251 Mercer Street, New York, NY 10012

Modifying Two-Phase Locking to Improve Performance

by

Theodore Johnson

Ultracomputer Note #133
April, 1988

This work was supported in part by the Applied Mathematical Sciences Program of the US Department of Energy under contract DE-AC02-76ER03077 and in part by contract number W-7405-ENG-48.

Abstract

Studies of concurrency control performance suggest that Two-Phase Locking (2PL) suffers from data-contention thrashing because transaction blocking is uncontrolled, allowing blocked transactions to block other transactions. Our algorithm improves the performance of 2PL in certain situations by aborting blocked transactions that block other transactions. This paper presents an analysis that validates this claim. The new algorithm gives better performance if aborting transactions is not too expensive.

1 Introduction

Concurrency control is necessary for the correct processing of transactions, but results in a lower throughput as some transactions will have to be aborted or held back to ensure correctness.

Concurrency control mechanisms are classified as being *optimistic* or *pessimistic* [BG81]. Pessimistic concurrency control ensures correctness by blocking transactions when further progress might cause a non-serializable execution. Optimistic concurrency control aborts non-serializable transactions. Optimistic concurrency control is so named because it was designed for the 'optimistic' situations when conflicts are rare. Two-phase locking (2PL) is the most common pessimistic technique, and timestamping and certification are the most common optimistic techniques.

Studies of concurrency control performance show that optimistic and pessimistic concurrency control are mis-labeled. [PR83] and [ACL87] show that, in the absence of resource contention, 2PL performs better than certification if data access conflicts are rare, but certification will give a higher throughput as data access conflicts become more common. [FR85] derives asymptotic bounds that predict this phenomenon. [TGS85] shows that 2PL is susceptable to data-contention thrashing. Because certification causes aborts, it results in a higher load on the underlying computer system's resources. If resource contention is significant, [ACL87] shows that 2PL always gives a higher throughput.

2 The Algorithm

With 2PL, a blocked transaction may block other transactions. With certification, an aborted transaction will not cause another transaction to abort. [FR85] defines *essential blocking* to be the property of a concurrency control algorithm that does not let blocked (aborted) transactions block (abort) other transactions. It is for this reason that certification gives a higher throughput than 2PL if data-contention is significant. However if data conflicts are rare, then 2PL gives a higher throughput. Thus, if few transactions are blocked by blocked transactions, it is better to let the blocked transactions wait.

The problem with 2PL is that when data-contention becomes significant, the waits-for trees (assuming that deadlock is immediately resolved) may become large and deep. This motivates the following family of concurrency control algorithms: whenever a waits-for tree passes a threshold on its size, a heuristic is run to prune the tree (abort some of the blocked transactions). The hope is to find a heuristic that prevents excessive blocking without causing too many aborts.

In this paper we examine an analytically tractable algorithm. The pruning heuristic is as follows: a waiting transaction is aborted whenever there is a path is the waits-for tree from the root to a leaf involving three or more transactions. Aborts occur only when a transaction becomes blocked and adds to a waits-for tree. After pruning, the waits-for trees are only one or two transactions deep. Therefore, there are only two cases to consider when pruning waits-for trees: when the path contains three and four transactions. We call this algorithm *Two-Deep 2PL*. Figure 1 shows the actions to take in both circumstances.

3 The Analysis

The analysis of Two-Deep 2PL is inspired by the analysis of 2PL in [TGS85]. Figure 2 shows the model of transaction processing. The analysis does not use stochastic processes, rather it is a mean-value analysis. The arrows in the figure indicate the flow of transactions, not state probabilities. The analysis uses Little's law and the principle of operational flow balance to derive its results.

We define:

D number of data granules

N number of concurrent transactions (blocked and active)

k number of data granules locked by each transaction

T mean inter-request time

w time to complete one stage of processing in the absence of resource contention

$\tau(N_e)$ processing time expansion factor due to resource contention.

B waiting time for blocked transactions

N_i number of active transactions holding i locks

W_i number of blocked transactions that hold i locks

N_L number of locks held

N_e number of executing transactions

N_b number of blocked transactions

b_i blocking rate at stage i

a_i restart rate at stage i

a abort rate, $\doteq \sum_{i=0}^{k-1} a_i$

c_i rate at which transactions enter stage i

p_c probability of conflict at each stage

p_r probability of restart at each stage

In the model of transaction processing, we ignore aborts due to deadlock. Two-Deep is a deadlock-free protocol. Also, [TGS85] ignores aborts due to deadlock, and we intend to compare our analysis to their analysis. We assume that the number of data granules that each transaction accesses is known. Each transaction accesses the same number of data granules. Data granule access is random and uniform; each data granule access is independent. When a transaction restarts, another one takes its place to keep N constant. The effect is that aborted transactions are discarded. This is not a problem because the transactions are homogeneous.

The distribution of T is not important because we only need its mean value. Resource contention is factored out into a *resource-contention expansion factor*, $\tau(N_e)$, so that $T = w\tau(N_e)$. This approximation makes the solution of the transaction processing system independent of the underlying computer system. The inter-request time factors out in the calculation, avoiding another level of iteration. We make the assumption that p_c and p_r are the same at each stage. Additionally, we assume that the waiting time for blocked transactions, B, is the same at each stage.

The rate at which transactions enter stage N_i is c_i. Using job flow balance, we get the job flow relation:

$$c_{i+1} = c_i - a_i$$

We also have the relationship $a_i = p_r c_i$. This gives us the recursive equation

$$c_{i+1} = c_i(1 - p_r) = c_0(1 - p_r)^{i+1}$$

By Little's law,

$$N_i = c_i T = c_0(1 - p_r)^i T = N_0(1 - p_r)^i$$

Using $b_i = p_c c_i$ and Little's law, we get

$$W_i = c_i p_c B = N_0(1 - p_r)^k p_c \frac{B}{T}$$

We now need an estimate for B. We note that if a transaction is blocked by another transaction, then the blocked transaction does not have to wait if the other transaction becomes blocked, as an abort will occur. Also, a transaction cannot become blocked by a blocked transaction, as again an abort will occur. Let us assume that each stage N_i has about the same number of transactions in it. If we weight each stage by i (the number of locks transactions at that stage hold) to use as a probability, (Pr[blocking transaction at stage i] $= i / \sum_{j=0}^{k} j$) then take the expected stage that the blocking transaction is at, the result is that $B = \frac{k}{3}T$.

Several simplifying assumptions are involved in the tractable formula $B = \frac{k}{3}T$. On the one hand, the possibility that some other transaction may be in the queue for the data item the transaction is waiting for is ignored. However, the possibility that the blocking transaction is aborted is also ignored. Hopefully, the two assumptions cancel each other out. Using the formula for B in the formula for W_i, we get

$$W_i = N_0 p_c \frac{k}{3}(1 - p_r)^k$$

If we define N_e as the number of executing transactions and N_b as the number of blocked transactions, then

$$N_e = \sum_{i=0}^{k} N_i = \sum_{i=0}^{k} N_0(1 - p_r)^i = N_0 \frac{1 - (1 - p_r)^{k+1}}{p_r}$$

similarly,

$$N_b = \sum_{i=0}^{k-1} W_i = N_0 \frac{k}{3} p_c \frac{1 - (1 - p_r)^k}{p_r}$$

The number of concurrent transactions is the sum of the active and blocked transactions

$$N = N_e + N_b = \frac{N_0}{p_r}(1 - (1 - p_r)^{k+1} + p_c \frac{k}{3}(1 - (1 - p_r)^k))$$

The probability of conflict is $p_c = N_L / D$. We need to find the number of locks held in the system

$$\begin{aligned} N_L &= \textstyle\sum_{i=0}^{k} i N_i + \sum_{i=0}^{k-1} i W_i \\ &= N_0 \textstyle\sum_{i=0}^{k} i(1 - p_r)^i + N_0 p_c \frac{k}{3} \sum_{i=0}^{k-1} i(1 - p_r)^i \\ &= \frac{N_0}{p_r^2}(k(1 - p_r)^{k+2} - (k + 1)(1 - p_r)^{k+1} + (1 - p_r) + p_c \frac{k}{3}((k - 1)(1 - p_r)^{k+1} - k(1 - p_r)^k + (1 - p_r))) \end{aligned}$$

We next need to calculate the probability of restart. A restart occurs only if a transaction blocks. Let R be the event that a restart is caused and let C be the event that the blocked transaction has

children (blocks transactions). $\bar{C}$ is the event that the blocked transaction has no children. If the blocked transaction has no children, then an abort will be caused only if the transaction is blocked by a blocked transaction.

$$\begin{aligned}\Pr(R \wedge \bar{C}) &= \Pr(\text{conflict}) \Pr(\text{wait for waiting transaction}) \\ &= p_c(1 - N_e/N)\end{aligned}$$

If the blocked transaction has children, then an abort will be caused.

$$\Pr(R \wedge C) = p_c$$

The probability of restart is the sum of these two possibilities

$$\begin{aligned}p_r &= \Pr(R \wedge \bar{C}) \Pr(\bar{C}) + \Pr(R \wedge C) \Pr(C) \\ &= p_c(1 - N_e/N) \Pr(\bar{C}) + p_c(1 - \Pr(\bar{C})) \\ &= p_c(1 - \frac{N_e}{N} \Pr(\bar{C}))\end{aligned}$$

If we assume that transactions randomly and uniformly block other transactions, then the probability of having no children is probability of failing to block all of the blocked transactions

$$\Pr(\bar{C}) = (1 - 1/N_e)^{N_b}$$

so that

$$p_r = p_c(1 - \frac{N_e}{N}(1 - 1/N_e)^{N_b})$$

We now have three variables (N_0, p_c and p_r) and three equations relating them. Considering the polynomials involving p_r, we have little hope of finding an algebraic solution to the system of equations. Instead, we will find an iterative solution. First, find N_0 as a function of the parameters N and K, and of the variables p_c and p_r. Inverting the formula for N_t,

$$N_0 = \frac{Np_r}{1 - (1 - p_r)^{k+1} + p_c\frac{k}{3}(1 - (1 - p_r)^k)}$$

We next find p_c as a function of the parameters and p_r using $p_c = N_L/D$ and the formula for N_0

$$p_c = \frac{N(k(1-p_r)^{k+2} - (k+1)(1-p_r)^{k+1} + (1-p_r) + p_c\frac{k}{3}((k-1)(1-p_r)^{k+1} - k(1-p_r)^k + (1-p_r)))}{Dp_r(1 - (1-p_r)^{k+1} + p_c\frac{k}{3}(1 - (1-p_r)^k))}$$

After some algebraic manipulation, this becomes the quadratic equation

$$\begin{aligned}&p_c^2(Dp_r\tfrac{k}{3}(1 - (1-p_r)^k)) \\ &+p_c(Dp_r(1 - (1-p_r)^{k+1}) - \tfrac{Nk}{3}((k-1)(1-p_r)^{k+1} - k(1-p_r)^k + (1-p_r))) \\ &-N(k(1-p_r)^{k+2} - (k+1)(1-p_r)^{k+1} + (1-p_r)) = 0\end{aligned}$$

Given the input parameters N, k and D, and an input guess p_r, we can solve for p_c, then N_0, then an output p_r. The output p_r will be a strictly decreasing function of the input p_r, so we have a rapidly converging iterative solution.

Having found p_r, p_c, and N_0, we can find the performance measures of the transaction processing system.

$$N_k = N_0(1 - p_r)^k$$

$$N_e = \frac{N_0}{p_r}(1-(1-p_r)^{k+1})$$

$$a = \sum_{i=0}^{k-1} a_i = \sum_{i=0}^{k-1} \frac{N_0}{T} p_r (1-p_r)^i$$
$$= \frac{N_0(1-(1-p_r)^k)}{T}$$

$$t = \frac{N_k}{T}$$

Where T may be a function of N_e.

4 Comparison

To compare the new algorithm against 2PL, we will the formulae derived in [TGS85]. Find N_e, which is the only real root of:

$$-2(2k-1)x^3 - (4k^2N - 13kN + 6N - 6Dk)x^2$$
$$+2N(4k^2N - 7kN + 3N - 3Dk)x + (k-2)(k-2)^2N^3 = 0$$

Then,

$$t = \frac{N_e}{(k+1)T}$$

Figures 3 - 5 show the throughput for the Two-Deep 2PL and the throughput for 2PL plotted against N. Resource contention is ignored in these graphs. Also shown is the abort rate for the Two-Deep 2PL and the deadlock rate for 2PL. The deadlock rate for 2PL is calculated using the formulae given in [TGS85]. Tables 1 - 3 give the data used in the graphs. The Two-Deep 2PL gives substantially higher throughput, but the abort rate can become significant. The 2PL deadlock rate ranges from 1% of the Two-Deep abort rate for $k = 10$ to 10% of the abort rate for $k = 30$.

Figures 6 - 8 show the case of $k = 30$ plotted for a range of resource-contention expansion factors. These graphs are pessimistic because they do not consider the abort rate for 2PL. The expansion factor $r(N_e) = N_e/5 + 1$ models a computer system that becomes loaded to capacity with 5 transactions, which is reasonable for a computer system with I/O overlap. Figures 7 and 8 show the throughput for computer systems that become loaded to capacity with 10 and 20 transactions respectively. These expansion factors can represent the processing of interactive transactions, or using a parallel or distributed computer system. In figure 6, 2PL has a higher throughput, but Two-Deep 2PL is not as sucseptable to data contention In figure 7, Two-Deep 2PL eventually achieves a higher maximum throughput and is not susceptable to data contention. In figure 8, Two-Deep 2PL always has an equivalent or higher throughput than 2PL.

5 Implementation Notes

If a tree-pruning 2PL algorithm is to be practical, the tree-pruning heuristic must be easy to implement. We will assume that when a transaction is blocked, it can easily determine the identity of the blocking transaction. In addition, we will assume that the blocked transaction can easily determine whether the blocking transaction is itself blocked. These are safe assumptions, as this information must be kept by the lock table of a transaction processing system, and is easily accessible in a centralized, tightly-coupled or distributed system

In addition to the above, each transaction keeps a *wait count*, which counts the number of transactions that the transaction blocks. When a transaction blocks, it increments the wait count of the blocking transaction. When a transaction aborts, it decrements the blocking transaction's wait count.

When a transaction blocks, it checks its wait count. If the wait count is non-zero, the transaction aborts. If the wait count is zero, the transaction tests to see if the blocking transaction is blocked. If so, the blocking transaction is aborted. Otherwise, the blocked transaction is on a path that contains only two transactions, so the blocked transaction waits.

Two-Deep 2PL is inherently deadlock-free, as any deadlocks are detected and resolved by the tree-pruning heuristic. While extra overhead must be performed to run the tree-pruning algorithm every time a transaction blocks, deadlock detection does not need to be performed.

6 Conclusions

A simple tree-pruning variant of 2PL has been discussed and analyzed. Although the hueristic was chosen for analytical simplicity, the algorithm gives a higher throughput than 2PL if resource contention is not considered. The abort rate may be significant with the new algorithm. If the cost of aborting transactions is too high, then the new algorithm will result in poorer performance.

Two-Deep 2PL is does not abort transactions because of deadlock, while 2PL does. The deadlock rate for 2PL as predicted by [TGS85] is about 10% of the abort rate of Two-Deep 2PL for $k = 30$. The 2PL rate of deadlock becomes an increasing percentage of the Two-Deep abort rate as k increases. This comaprison suggests that Two-Deep might be especially useful for long transactions.

Other tree-pruning heuristics are possible. One example is pruning the waits-for tree whenever there is a path with *four* or more transactions. Such an algorithm might give a lower throughput than Two-Deep 2PL, but also a lower restart rate. Unfortunately, these algorithms are more difficult to analyze.

References

[ACL87] R. Agrawal, M. Carey, M. Linvey 'Concurrency Control Performance Modeling: Alternatives and Implications' *ACM TODS 12.2* 1987 pp. 609-654

[BG81] P. Bernstein, N. Goodman 'Concurrency Control in Distributed Databases' *Computing Surveys 13.2* 1981 pp. 185-221

[FR85] P. Franaszek, J. Robinson 'Limitations of Concurrency in Transaction Processing' *ACM TODS 10.1* 1985 pp. 1-28

[PR83] P. Peinl, A. Reuter 'Empirical Comparison of Concurrency Control Schemes' *Proceedings of the Ninth Int'l Conference on Very Large Data Bases* 1983 pp. 97-108

[TGS85] Y. Tay, N. Goodman, R. Suri 'Locking Performance in Centralized Databases' *ACM TODS 10.4* 1985 pp. 415-462

Number of Transactions	Two-Deep		2PL	
	throughput	abort rate	throughput	deadlock rate
20	1.76	.0104	1.76	.0003
40	3.39	.0764	3.37	.0012
60	4.89	.2387	4.84	.0027
80	6.24	.5429	6.13	.0052
100	7.46	.9520	7.21	.0084
120	8.54	1.530	8.02	.0129
140	9.50	2.260	8.44	.0190
160	10.3	3.147	8.07	.0283
180	11.1	4.182	-	-
200	11.7	5.36	-	-

Table 1: k=10

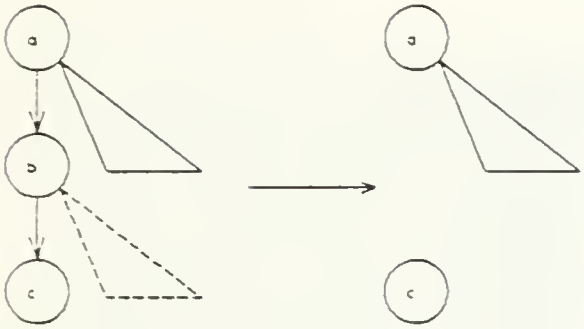
Number of Transactions	Two-Deep		2PL	
	throughput	abort rate	throughput	deadlock rate
5	.230	.0015	.230	.0001
10	.442	.0106	.442	.0006
15	.636	.0331	.634	.0014
20	.811	.0724	.802	.0027
25	.976	.1307	.941	.0045
30	1.10	.2094	1.04	.0070
35	1.23	.3088	1.08	.0105
40	1.33	.4285	.987	.0163
45	1.42	.5681	-	-
50	1.50	.7268	-	-

Table 2: k=20

Number of Transactions	Two-Deep		2PL	
	throughput	abort rate	throughput	deadlock rate
3	.092	.0011	.092	.0001
6	.174	.0078	.174	.0008
9	.246	.0232	.244	.0020
12	.307	.0493	.297	.0038
15	.358	.0868	.324	.0067
18	.401	.1355	.298	.118
21	.435	.1951	-	-
24	.463	.2650	-	-
27	.485	.3445	-	-
30	.502	.4325	-	-

Table 3: k=30

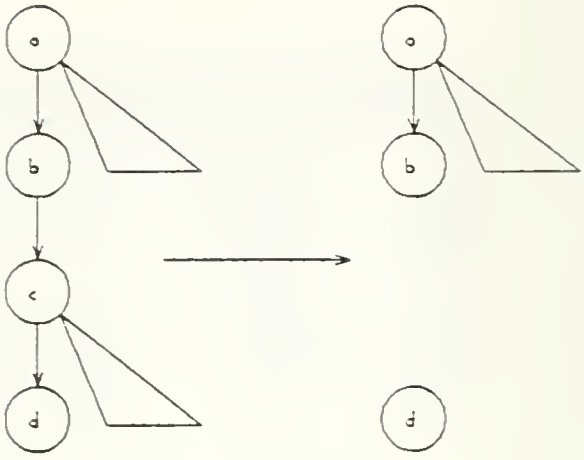
A) Three Transactions



Abort transaction b.

(If c was blocked by b, then b might have had children.)

B) Four Transactions



Abort transaction c.

Figure 1.

Figure 2.

Model of Transaction Processing

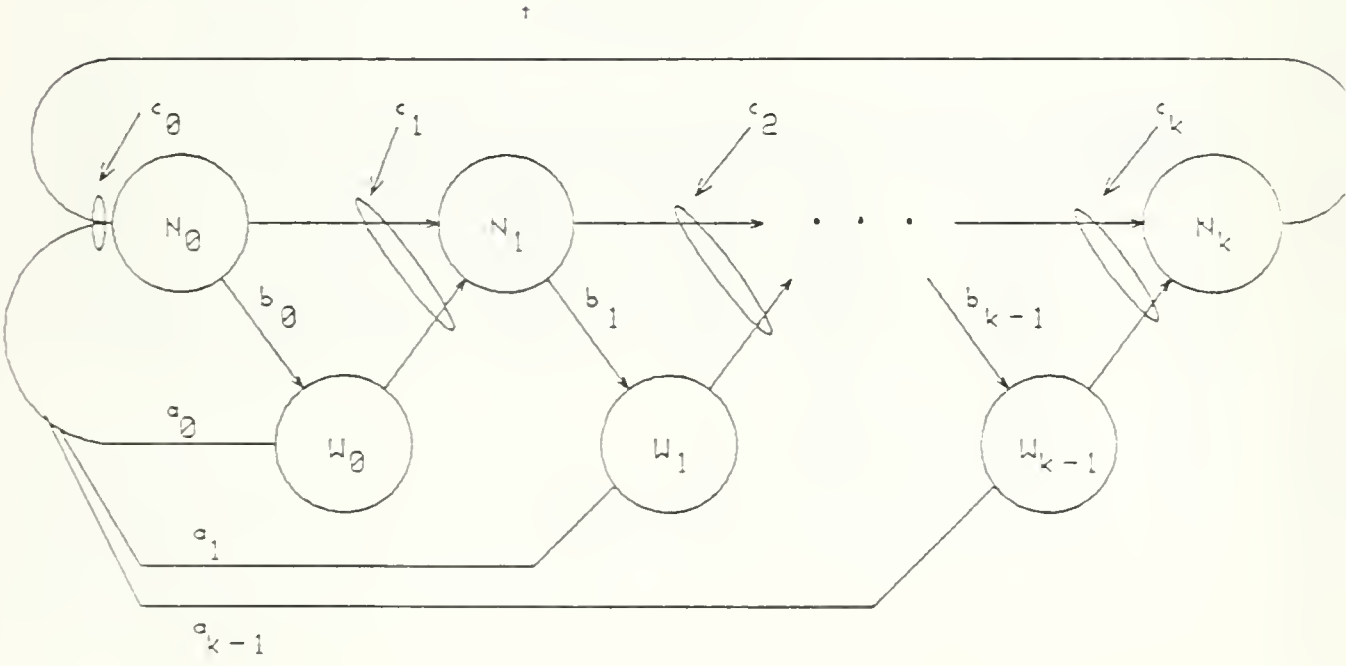


Figure 3.

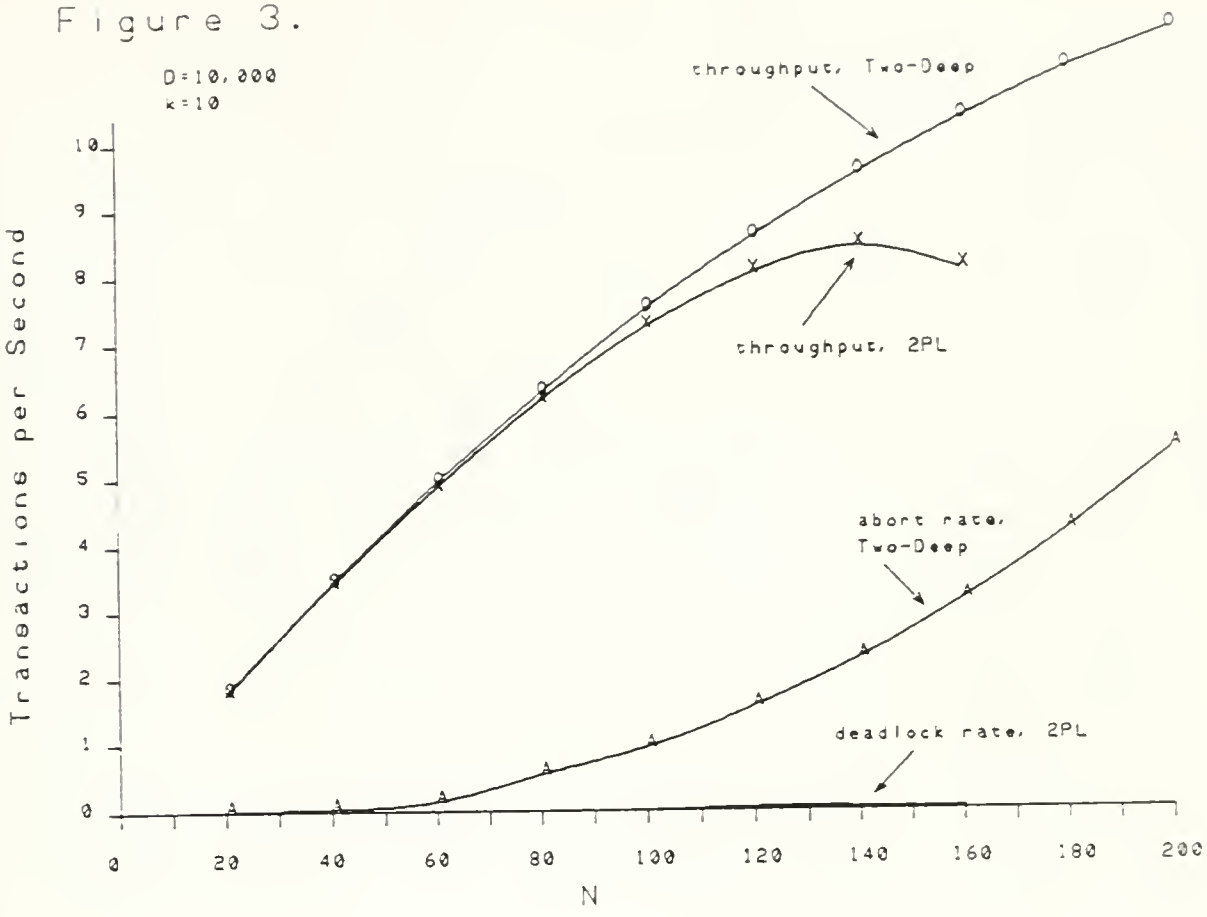


Figure 4.

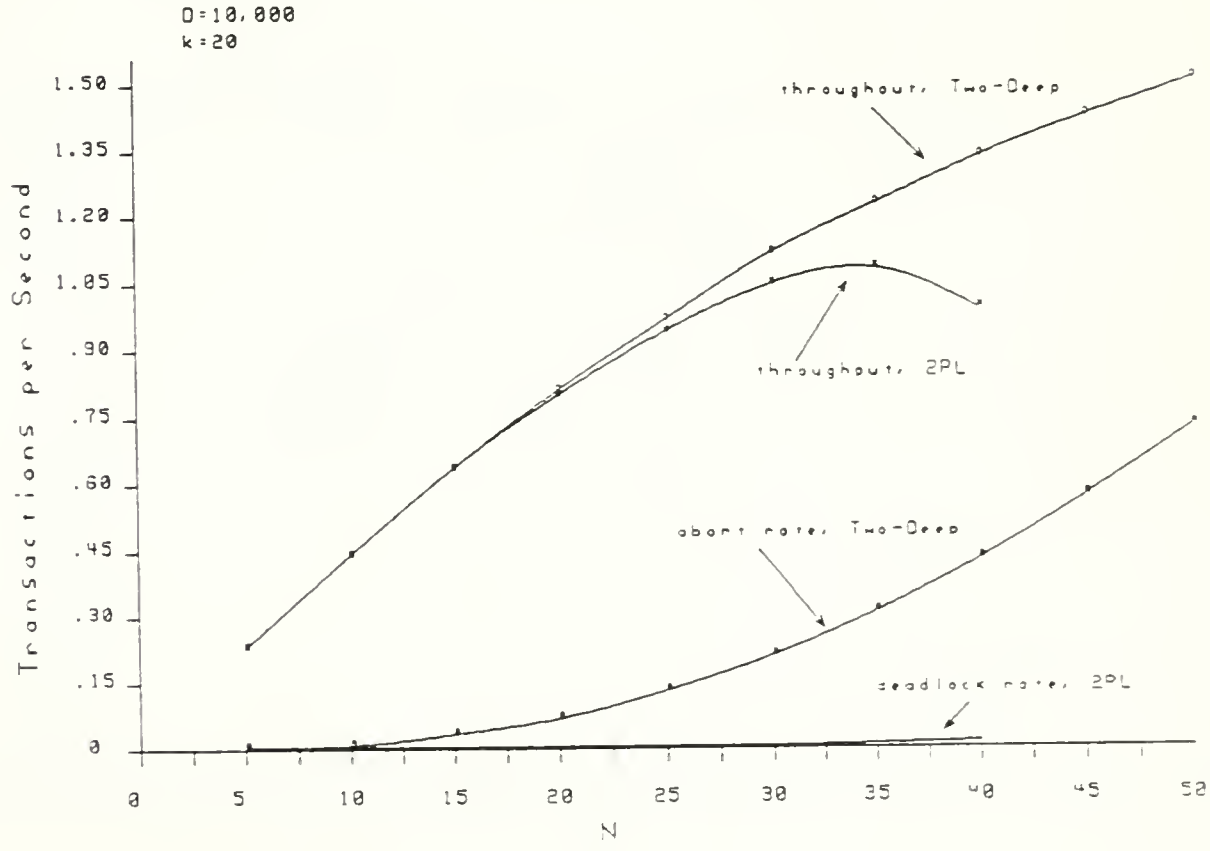


Figure 5.

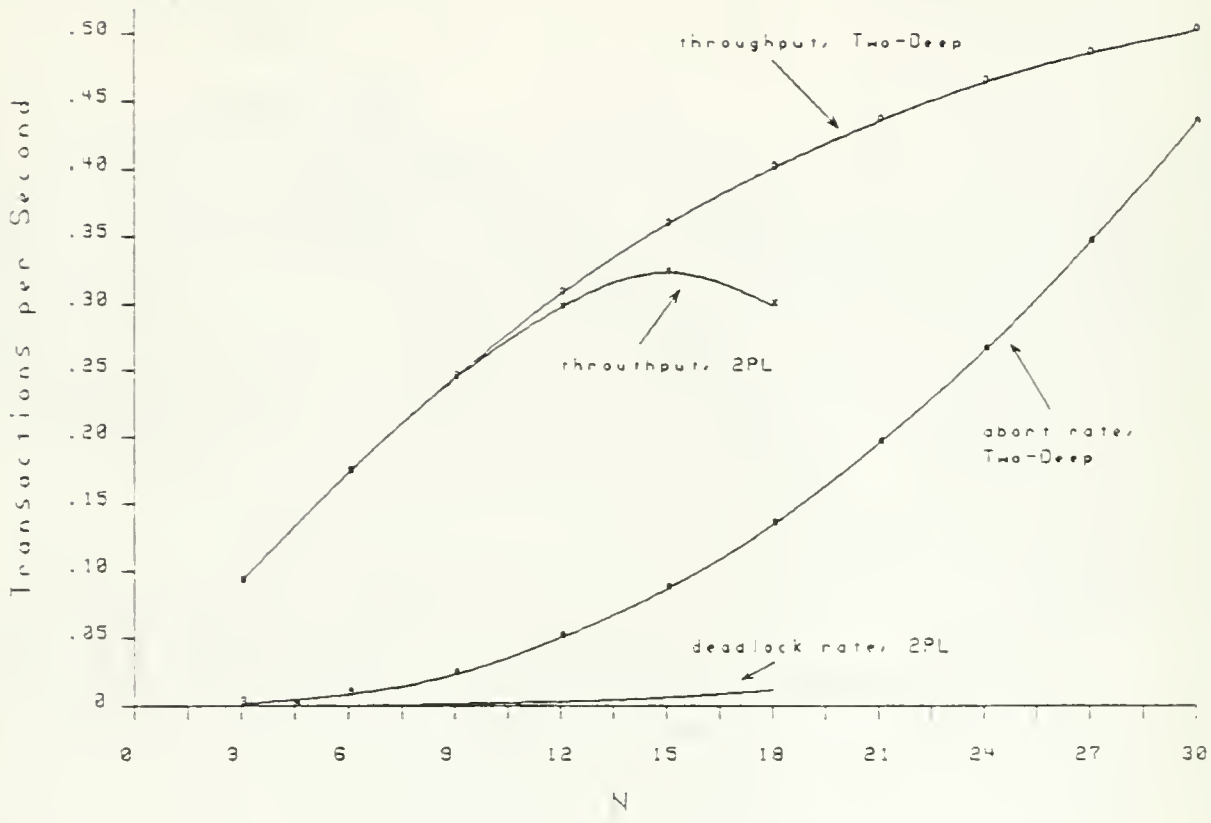


Figure 6.

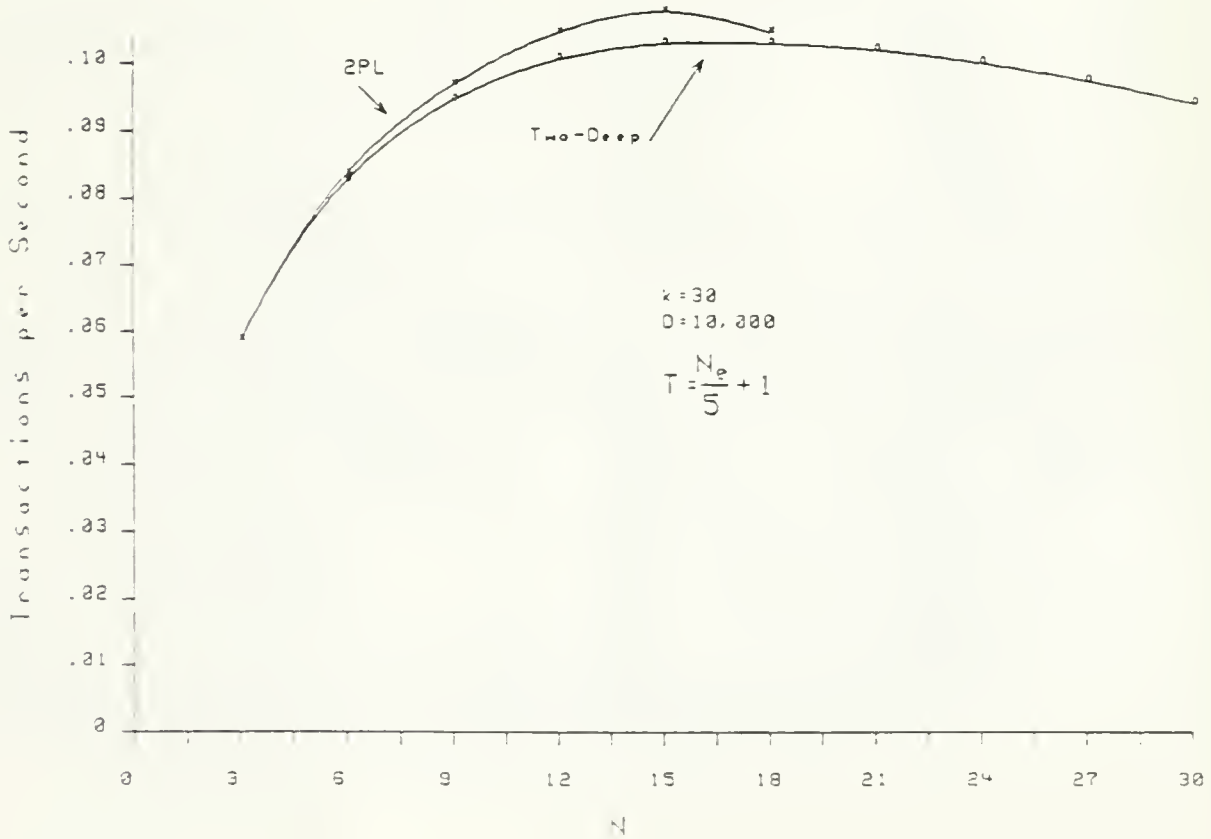


Figure 7.

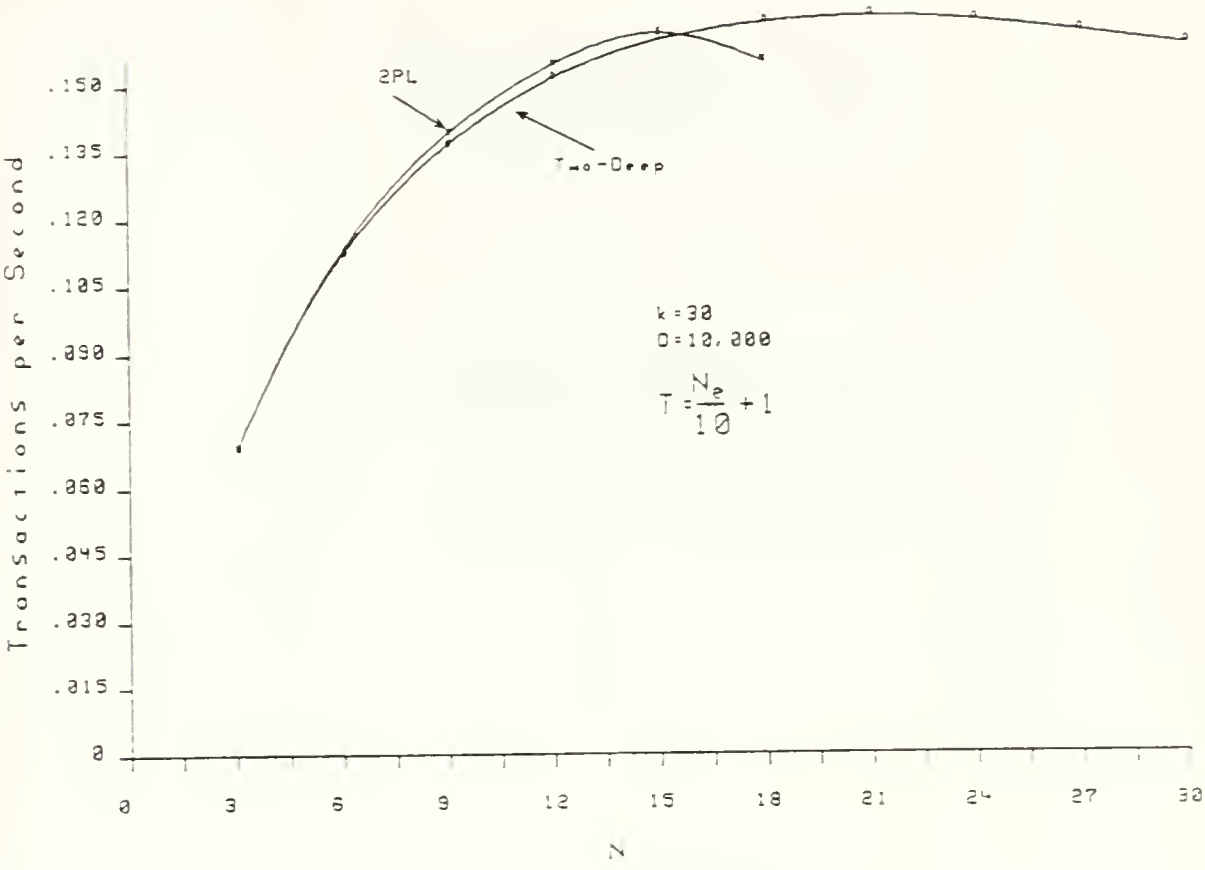
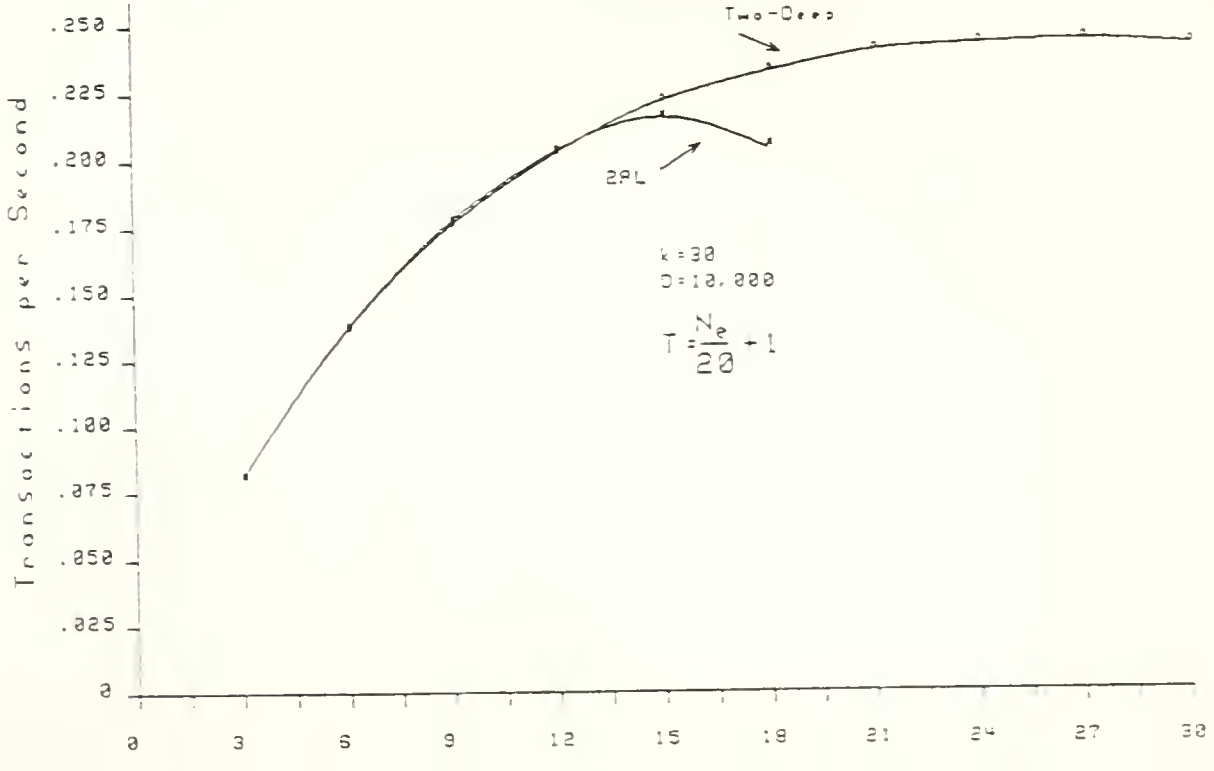


Figure 8.



NYU UCN-133
Johnson, Theodore
Modifying two-phase locking
to improve performance.
c.1

NYU UCN-133
Johnson, Theodore
Modifying two-phase locking
to improve performance.
c.1

DATE DUE	BORROWER'S NAME

LIBRARY
N.Y.U. Courant Institute of
Mathematical Sciences
251 Mercer St.
New York, N. Y. 10012